Spatial Locality Aware Disk Scheduling in Virtualized Environment

Xiao Ling, Shadi Ibrahim, Song Wu, Hai Jin

Spatial Locality Aware Disk Scheduling in Virtualized Environment

Xiao Ling, Shadi Ibrahim, Song Wu, *Member, IEEE,* Hai Jin, *Senior Member, IEEE*

Abstract—Exploiting spatial locality, a key technique for improving disk I/O utilization and performance, faces additional challenges in the virtualized cloud because of the transparency feature of virtualization. This paper contributes a novel disk I/O scheduling framework, named Pregather, to improve disk I/O efficiency through exposure and exploitation of the special spatial locality in the virtualized environment, thereby improving the performance of disk-intensive applications without harming the transparency feature of virtualization. The key idea behind *Pregather* is to implement an intelligent model to predict the access regularity of spatial locality for each VM. Moreover, *Pregather* embraces an adaptive time slice allocation scheme to further reduce the resource contention and ensure fairness among VMs. We implement the *Pregather* disk scheduling framework and perform extensive experiments that involve multiple simultaneous applications of both synthetic benchmarks and MapReduce applications on Xen-based platforms. Our experiments demonstrate the accuracy of our prediction model and indicate that *Pregather* results in the high disk spatial locality and a significant improvement in disk throughput and application performance.

Index Terms—Virtualization, disk-intensive, I/O scheduling, spatial locality, efficiency

1 Introduction

Virtualization technology is extensively leveraged in cloud environments: it enables multiple *virtual machines* (VMs) — with multiple operating systems and applications — to run within a physical server. For example, Amazon web services [1] rely on the Xen virtualization hypervisor to provide the VM-based infrastructure as a service (IaaS) solution, which enables users to lease and customize their environments in order to run their applications. Virtualization however imposes new challenges in the scheduling and the allocation of system resources. With the volume of data growing rapidly and multiple disk intensive applications with different disk I/O characteristics (mixed applications) sharing an infrastructure [2], [3], allocating disk resources efficiently while guaranteeing VMs' I/O performance for preserving a high disk throughput becomes of key importance in virtualized environments.

Exploiting spatial locality is an important technique for scheduling I/O requests to improve disk I/O efficiency (*i.e.*, high spatial locality results in a significant reduction in disk seek delay and rotation overhead, which leads to high disk throughput). For example, traditional file systems often allocate the accessed data of a process as contiguous blocks if possible, so disk scheduling can easily, according to I/O characteristics of the process, exploit spatial locality of requests. Unlike traditional environments, in a virtualized environment achieving high spatial locality is a challenging task, due to the *transparency feature of virtualization* which causes the semantic gap isolation between the hypervisor and guest VMs. As a result, when VMs with different disk intensive applications share disks in the cloud environment, the block I/O layer lacks a global view of the I/O access patterns of processes [4]. The lack of coordination between file systems in both the hypervisor and VMs reduces the efficiency of exploiting disk locality in virtualized environments. Moreover, a VM leased by users may encapsulates more than one application (*e.g.*, multiple applications, including file-editing and media streaming, run in a virtual desktop; a Hadoop application generates multiple Map processes to parallel access data), which in turn increases the complexity and irregularity of the I/O behavior of the VM.

Research effort has been directed toward improving disk efficiency through exploiting spatial locality in virtualized environments. These efforts use either *invasive mode scheduling* (*i.e.*, select the disk pair schedulers within both the hypervisor and the VM according to the applications' access patterns [5], [6]), or *non-invasive mode scheduling* (*i.e.*, schedule the I/O requests while treating a VM as a black box [7], [8]). However, the aforementioned solutions target similar types of applications (mainly read-dominated applications), and cannot be applied when a VM encapsulates mixed applications [4], [9]. Moreover, they come at the cost of violating the transparency feature of virtualization. Besides, although these solutions exploit spatial locality to improve disk I/O utilization, they may not ensure I/O performance of each VM. For example, when some VMs deployed applications with low spatial locality and some VMs

- *X. Ling, S. Wu and H. Jin is with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China. Song Wu is the corresponding author. Email:wusong@hust.edu.cn.*
- *X. Ling is also with Information & Communication Company of Hunan Electric Power Corporation of State Grid in China.*
- *S. Ibrahim is with Inria, Rennes Bretagne Atlantique Research Center, France.*

deployed applications with strong spatial locality run together, they prefer to serve applications with strong spatial locality with cost of the performance of the applications with low spatial locality. This may lead to starving some applications with low spatial locality and I/O contention among VMs.

This paper follows this line of research and contributes to the goal of improving throughput for complex I/O workloads, including write-dominated applications or mixed applications, by enabling efficient disk utilization and guaranteeing I/O performance of VMs, with preserving the transparency feature of virtualization. To end this, we solve two key issues: (1) detecting spatial locality in virtualized environments without any prior information of workloads; (2) making full use of spatial locality while guaranteeing I/O performance of each VM. In this paper we make the following three contributions to achieve this goal:

1) We investigate the spatial locality of disk data accesses in virtualized environments. By tracing the I/O requests of VMs with mixed applications, we observe that the disk data accesses are grouped into *regions*, bounded by the virtual disk sizes, and within each region the disk data accesses are grouped into *sub-regions*, which correspond to the applications' access patterns.
2) We introduce an intelligent prediction model that uses a temporal access-density clustering algorithm to analyze the data access of a VM with mixed applications. Our model can predict the distribution of sub-regions with spatial locality within each region and the arrival times of future requests accessing these sub-regions (*i.e.*, detect the sub-regional spatial locality for each VM).
3) We propose *Pregather*, a disk scheduling framework with a spatial-locality-aware heuristic algorithm in the hypervisor for exploiting the *special* spatial locality (*i.e.*, the regional and sub-regional spatial locality) to reduce disk seek and rotational overhead in the virtualized environment: *Pregather* does that — thanks to our prediction model — *without any prior knowledge* of the applications' access patterns. Besides, *Pregather* embraces an adaptive time slice allocation scheme based on the special spatial locality of VMs to further ensure fairness among VMs while improving disk I/O utilization.

We build the prototype of *Pregather* in Xen. Our evaluations, using synthetic benchmarks, a MapReduce application (distributed sort) and the database workloads, demonstrate the accuracy of the intelligent prediction model and the throughput benefits from our approach: *Pregather* achieves high disk spatial locality upon ensuring I/O performance of VMs, and thus improves the disk utilization and the applications' performance. For example, when multiple VMs with mixed applications runs together, in contrast to the default Xen disk I/O scheduler -*Completely Fair Queuing* (CFQ), *Pregather* achieves throughput performance improvement by a factor of 1.5x and improves I/O performance of applications.

The rest of the paper is organized as follows. Section 2 observes the disk access patterns in virtualized environments with mixed applications. Section 3 discusses our prediction model. Section 4 describes the design and implementation of *Pregather*. Section 5 details the performance evaluation. Section 6 discusses the related work. Finally, we conclude the paper in Section 7.

2 Observing Disk Access Patterns in Virtualized Environments

In this section, we seek to obtain an overview of understanding the spatial locality in virtualized environments. Ideally, we would like to get a rough idea of the disk access patterns in virtualized environments, and the impact of both virtualization features and mixed applications on the access patterns. So we experiment with two typical scenarios — typical synthetic benchmarks running on VMs and a MapReduce application running on a virtual cluster — and trace activities of I/O requests in the two scenarios.

2.1 Experimental Setup

In the synthetic-benchmarks scenario, four guest VMs with different mixed I/O workloads run in one physical node. Main features of access patterns of I/O workloads include read, write, sequential and random access. As shown in Table 1, we use *sysbench* [10] to generate these workloads with different disk access patterns and deploy them in different VMs.

In order to further observe disk access pattern in the real intensive-applications scenario, we deploy Hadoop (Hadoop-0.20.2[1]) on a thirteen-nodes virtual cluster running on a three-nodes physical cluster, and run the sort benchmark[2]. Note that Hadoop with the sort benchmark is a typical and widely used example of the real I/O intensive and parallel applications. The access patterns of processes of sort benchmark are complicated due to the mixing of read, write, sequential and random access and the variable access intervals. We deploy five VMs in physical machine 1 (PM1), and four VMs in PM2 and PM3, respectively. In each PM, four VMs as data nodes owns eight map processes. The data set is 6GB for the sort benchmark (64MB block size).

In the two scenarios, the physical node is equipped with four quad-core 2.40GHz Xeon processor, 22GB of memory and one dedicated 1TB SATA disk of which availabe space is 100GB, running CentOS5 with kernel

1. Although the versions of Hadoop are changing, the Hadoop-0.20.2 is the most classical and stable version compared to other Hadoop versions. Besides, all Hadoop versions generate parallel processes (Map processes) to intensively access data and the types of their workloads are I/O intensive, such as sort, wordcount and so on.

2. In the sort benchmark, each mapper sorts the data locally, and each reducer merges the results from different mappers. And, the map input and the reduce output have the same data size as the input data. The I/O access patterns of processes are due to the sort benchmark and are not impacted by the changes of Hadoop versions

TABLE 1: **The description of the workloads running within VMs**

VM	Workload	Description
VM1	Sequential Read-M (*SR-M*)	16 threads sequentially read 128 files, with total size of 1 GB
	Sequential Read-S(*SR-S*)	16 threads sequentially read 1 file, with the size of 1 GB
VM2	Sequential Write-M (*SW-M*)	16 threads sequentially write 128 files, with total size of 1 GB
	Sequential Write-S(*SW-S*)	16 threads sequentially write 1 file, with the size of 1 GB
VM3	Sequential Read-M (*SR-M*)	16 threads sequentially read 128 files, with total size of 1 GB
	Random Read & Write (*RRW*)	16 threads randomly write and read 128 files, with total size of 1GB
VM4	Random Read (*RR*)	16 threads randomly read 128 files, with total size of 1GB
	Random Write (*RW*)	16 threads randomly write 128 files, with total size of 1 GB

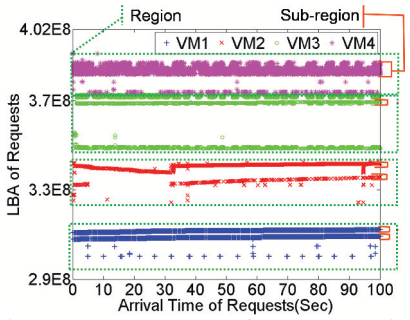
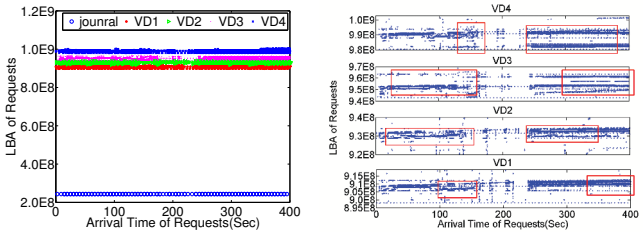


Fig. 1: **Disk access patterns of VMs in the synthetic-benchmarks scenario**



(a) The distribution of arriving requests in disk of PM2

(b) Zooming access patterns of each VM in PM2

Fig. 2: **Disk access patterns of VMs in the MapReduce-application scenario**

2.6.18. All results are obtained using Xen version 4.0.1 with Blktap AIO driver [11]. The guest VM is configured with two Virtual CPUs, 1GB memory and 12 GB virtual disk (the virtual disk is mapped to a default file-backed image). The default disk scheduler of VMs is Noop[3] while the default disk scheduler of the hypervisor is the default CFQ. Besides, we use the *blktrace* tool [12] to track the logical block addresses (LBAs) and arrival times of requests from VMs.

2.2 Synthetic-benchmarks Scenario

Fig. 1 shows the change in the LBAs of arriving requests from four VMs in the synthetic-benchmarks scenario.

Our *first observation* is that the LBAs of requests from different VMs can be grouped into different regions which are occupied by VM images. The ranges [3.0012×10^8-3.1739×10^8], [3.244×10^8-3.426×10^8], [3.4863×10^8-3.727×10^8] and [3.738×10^8-3.911×10^8] represent the regions of VM1, VM2, VM3 and VM4, respectively. The size of each region is smaller than the 12GB size of VM image. The reason is that the file system of hypervisor assigns contiguous disk blocks to the VM image in order to improve disk efficiency. Accordingly, the VM has regional spatial locality when the average seek distance of a VM is smaller than the size of its image.

Our *second observation* is that looking at the disk accesses within each VM, LBAs and arrival times of requests divide the region of each VM into several sub-regions over time. These sub-regions differ in their ranges and access frequencies. The reason is that the file system of a VM also assigns contiguous virtual blocks to the applications without obtaining physical disk characteristics, and accordingly all VM image formats (*e.g.*, RAW, Qcow [13], Fvd [14]) try to map LBAs of these virtual blocks into contiguous offsets in the VM image. As the hypervisor treats a VM as a process, the file system of the hypervisor maps the contiguous offsets into as contiguous LBAs as possible. Furthermore, because applications access their own data sub-regions, requests from the same application have sub-regional spatial locality, especially from applications with sequential access.

As shown in Fig. 1, these VMs have different characteristics of sub-regional spatial locality. This is because the features of access pattern of application impact on the sub-regional spatial locality of VMs. First, in contrast to VM1 and VM2, both VM3 and VM4 do not have a clear sub-regional spatial locality (*i.e.*, the ranges of observed sub-regions are larger), because of the random access of data sets exhibited by random applications. For example, during 100s the requests from VM1 are mainly concentrated into two narrow sub-regions similar to thin horizontal lines (both ranges of sub-regions are less than 1.4×10^6). Instead, the most of the requests from VM4 are distributed in a large sub-region randomly: around 3.845×10^8 with a maximum distance of $\mp 2.7 \times 10^6$. Second, for VMs with write applications, the distribution of sub-regions and their ranges are more diverse. Moreover, the access frequencies of these sub-regions are often changing. For instance, the distribution and access frequencies of sub-regions of VM2 during 0 to 30s are different from the ones after 30s. This is due to a non-deterministic allocation of the write data set, the impact of disk cache, and update of inodes of files.

Fig. 1 also shows that some accessed sub-regions own the low access frequencies at the beginning of each region, thus not having spatial locality. This can be

3. Noop has recently been used as the default VM scheduler because any (re)ordering at the VM level will be counterproductive as I/O requests from VMs will be dispatched to the disk device according to the disk scheduler at the hypervisor level. Thus, it is better to simply push the requests as early as possible, so as to save CPU cycles and avoid any conflict that could occur between the I/O schedulers at the hypervisor and VM levels.

explained due to: (1) writing log files — the journal process in a VM periodically writes the system log files, the position of log files are normally at the start of the VM image; and (2) updating the metadata of the VM image, such as the access time and the changes of file content. The position of the inode is generally at the start of the VM image. Moreover, applications with write or random access introduce the frequent update of log files and the metadata, thus resulting in a growing number of sub-regions with low access frequencies, such as VM3 and VM4 in Fig. 1.

2.3 MapReduce-application Scenario

To prove our observations further, we track the information of arriving requests in the block layers of physical machines when running the sort benchmark. Fig. 2 shows the distribution of LBAs of arriving requests in disk of PM2[4] during 400s.

First, Fig. 2(a) describes that most of requests are from the four VMs (named VD1, VD2, VD3 and VD4, respectively) and a few requests are from journal process in hypervisor. These requests from different VMs are grouped into different regions in the MapReduce-application scenario as well as in the synthetic-benchmarks Scenario. For example, the LBAs of requests from VD1, VD2, VD3 and VD4 concentrate on the ranges [8.981×10^8-9.194×10^8], [9.202×10^8-9.424×10^8], [9.429×10^8-9.688×10^8] and [9.803×10^8-1.001×10^9], respectively. Also, the sizes of these regions are smaller than the image file sizes of corresponding VMs. So VMs also own regional spatial locality in the MapReduce-application scenario. Besides, because the requests from journal process in the hypervisor access a fixed region periodically, this region does not have spatial locality.

Then, we amplify the regions corresponding to VMs with the I/O processes of Hadoop, in order to observe the sub-region spatial locality of VMs further. As shown in Fig. 2(b), within each region, LBAs of requests also concentrate different sub-regions with time. However, the distribution and ranges of VMs sub-regions are more variable and the number of sub-regions with low access frequency increases in the MapReduce-application scenario, compared to the synthetic-benchmarks scenario. The reason is that the VM as the date node of Hadoop encapsulates more than one I/O process whose access patterns interleave sequential and random access and are changing with time. These result in more complicated I/O access patterns of VMs. Meanwhile, the increasing number of write operations and random access brings in frequently updating log file and metadata, which causes the increasing number of sub-regions with low access frequencies (*e.g.*, VD1, VD3, and VD4).

Nevertheless, in some time periods each region also has some sub-regions which are similar to almost horizontal lines. As shown in Fig. 2(b), we use red rectangles to point out several horizontal lines within each VM's region during the time periods. This shows that a lot of requests in the time periods frequently access these sub-regions. Thus, the VMs with the MapReadue application own sub-regional spatial locality in these time periods. Moreover, the characteristics of sub-regional spatial locality of these VMs are changing with time. For instance, the distribution and access frequencies of sub-regions of VD2 during 0 to 50s are more diverse compared to ones during 50s to 100s. This is due to the change of access patterns of processes, especially write operations and random access. Furthermore, when the access patterns of some Hadoop's processes in a VM are sequential access, the thin lines exist within the region and the VM owns strong sub-regional spatial locality during these periods. For example, during a period from 20s to 150s, in spite of several sub-regions with low access frequencies or with larger ranges, the region of VD3 still owns two thin horizontal lines. This is because two processes with sequential access patterns access different data sets simultaneously although other processes randomly access data within the VD3's region during this period. Thus the regularity of sub-regional spatial locality of VM is changed based on access patterns of processes running in the VM.

2.4 Discussion

In summary, with respect to the cases when VMs with multiple processes run together, *we observe the special spatial locality in virtualized environments: regional spatial locality across VMs and sub-regional spatial locality between requests from the same VM*. Regional spatial locality is bounded by the size of VM image. Moreover, the sub-regional spatial locality of a VM is obvious when processes with sequential access run within a VM.

The virtualization transparency leads to inefficient exploitation of spatial locality and thus lowers disk utilization (*i.e.*, increases disk head seeks (movement) between sub-regions); the traditional non-work-conserving schedulers, including CFQ [15] and Anticipatory scheduler (AS) [16], are effective at preserving the spatial locality exhibited by individual processes, but treat a VM process simply as a general user process and never recognize the sub-regional spatial locality of a VM, resulting in the low spatial locality (*e.g.*, as shown in Section 5, the seek distance at zero is only 55% under CFQ in the synthetic-benchmarks scenario). The aim of our work is to make full use of the special spatial locality of VMs to improve physical disk efficiency and thus enhance the performance of applications in the virtualized environment.

According to above observations, our aim faces two mainly issues: (1) how to detect the regularity of the special spatial locality, especially sub-regional spatial locality, without any prior knowledge of applications; (2) how to maximize disk I/O utilization while guaranteeing I/O performance of VMs, especially when VMs with strong sub-regional spatial locality and VMs with weak sub-regional spatial locality share storage.

4. Because of limitation of pages, we show the distribution of LBA of requests in one of three physical machines. The observations of other two physical machines are same as that of PM2.

TABLE 2: Variables of the *vNavigator* model

Var.	Description	Var.	Description
VM	a guest VM	$\triangle n$	the number of requests accessing Z_j during interval $[T_r, T]$
P(R)	the LBA of a request *R*	U_i	the i^{th} sub-region unit with sub-regional spatial locality of a VM in current prediction window
T(R)	the arrival time of a request *R*	λ	a decay factor
B	the offset in the disk	$W(R_j, T)$	the contribution of R_j to spatial locality at prediction window T
Z_j	the j^{th} equal-sized zone whose size is *B*	$D(Z_j, T)$	the temporal access-density of Z_j at prediction window T
R_j	a request accessing Z_j	$\delta(VM, T)$	the temporal access-density threshold of a VM at prediction window T
R_j^m	the m^th request accessing Z_j	$ZT(Z_j, T(R_j))$	the average access time interval of Z_j when a request R_j access Z_j
T	the current prediction window	$SR(U_i)$	the range of U_i
T_r	a prediction window when R_j arrives	$ST(U_i)$	the future access interval of U_i

3 Prediction Model of the Locality and Regularity of Disk Accesses

As discussed in Section 2, the regional spatial locality can be easily observed according to the VM image size. But the sub-regional spatial locality cannot be observed in the virtualized environment due to the virtualization transparency. Consequently, the disk scheduler in the hypervisor cannot efficiently exploit the sub-regional spatial locality of the VM. To this end, we design an intelligent prediction model, named *vNavigator*, to predict the regularity of the sub-regional spatial locality of a VM(*i.e.*, the distribution of sub-regions with spatial locality and access intervals of these sub-regions). Hence the *vNavigator* model helps to guide I/O scheduling in the hypervisor. Based on the discussion about sub-regional spatial locality in Section 2, the design of *vNavigator* model faces three challenges: (1) the distribution of sub-regions with spatial locality is changing with time, and varies based on the access patterns of applications; (2) requests from background processes (named discrete requests) within a VM interfere with the prediction of future requests with sub-regional spatial locality; and (3) different sub-regions with spatial locality may have different access regularity. For clarity, Table 2 lists the variables used in the *vNavigator* model.

The *vNavigator* model uses a *temporal access-density clustering* (*TAC*) algorithm to analyze the historical data access within a VM image file to predict the future sub-regional spatial locality of the VM. As shown in Fig. 3, considering the frequent changes in both the range and regularity of the sub-regions with spatial locality, the *TAC* algorithm divides the disk space into a series of equal-sized *zones* (denoted by $Z = \{Z_1, Z_2, ..., Z_n\}, n = \frac{length(disk)}{B}$). To capture and predict the change of the spatial locality in time, the *TAC* algorithm also divides the allocated serving time of a VM into several equal time windows (*prediction window*). By tracking the arrival time and the LBAs of VM's requests, the *TAC* algorithm quantizes the zones' access frequencies in previous windows to estimate the spatial locality of zones in the current prediction window.

Zones with possible spatial locality draw out the distribution of the sub-regions with relative spatial locality: given that zones with similar spatial locality may have different access frequencies and access intervals, the *TAC* algorithm therefore groups, within the same prediction window, neighboring zones into larger units (sub-region units). The distribution of sub-regions with spatial locality can be represented as $U = \{U_1, U_2, ..., U_i\}, U_i \subseteqq VM$, where U_i represents the i^{th} sub-region unit with spatial locality. A sub-region unit may consist of one or two neighboring zones (further details are provided in the following subsections). Accordingly, the *vNavigator* model actually predicts the range of U_i and the arrival time interval of future requests accessing U_i.

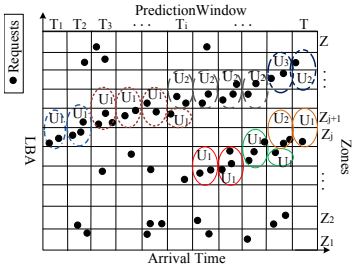


Fig. 3: ***vNavigator* model:** sub-region unit distribution

3.1 Quantization of Access Frequency

The *TAC* algorithm introduces a temporal access-density to quantize the access frequency of a zone. The temporal access-density of a zone is the sum of the contributions of historical requests to the future possibility of the zone's spatial locality in the current prediction window. Because the sub-regional spatial locality changes with time, the impact of recently arrived requests is greater than that of older requests on predicting sub-regional spatial locality. The contribution of historical requests therefore decline with time. Accordingly, we introduce an access weight of the request to represent the contribution of a request in the current prediction window. The access weight uses a decay factor (λ) to quantize the relationship between the contribution of the request and time.

Definition 1: The access weight of a request (R_j) to the spatial locality of a zone (Z_j) in the current prediction window is:

$$W(R_j, T) = \lambda^{-(T-T_r)} \tag{1}$$

where $\lambda > 1$ and R_j accesses Z_j in T_r.

According to (1), the access weight is 1 at the beginning, then decays toward zero with passing time. And the temporal access-density of the zone is defined as follows:

Definition 2: The temporal access-density of a zone (Z_j) is the sum of the access weights of the requests accessing

this zone in the current prediction window:

$$D(Z_j,T) = \sum_{P(R)\in Z_j} W(R,T) \quad (2)$$

where R represents all requests accessing Z_j until T.

According to (1) and (2), and to simplify the computational cost of the temporal access-density of a zone, we obtain the following *Lemma 1*.

Lemma 1: Given that $\triangle n$ is the number of arrived requests accessing Z_j between T_r and T ($T_r < T$), $D(Z_j,T)$ is given by:

$$D(Z_j,T) = \lambda^{-(T-T_r)} D(Z_j,T_r) + \triangle n. \quad (3)$$

Proof: n is the number of requests accessing Z_j until T, and R_j^k represents the k^{th} request to access Z_j.

$$\begin{aligned} D(Z_j,T) &= \sum_{k=1}^{n+\Delta n} W(R_j^k,T) = \sum_{k=1}^{n} W(R_j^k,T) + \sum_{k=n+1}^{n+\Delta n} W(R_j^k,T) \\ &= \lambda^{-(T-T_r)} \sum_{k=1}^{n} W(R_j^k,T_r) + \sum_{k=n+1}^{n+\Delta n} \lambda^{T-T} \\ &= \lambda^{-(T-T_r)} D(Z_j,T_r) + \triangle n \end{aligned}$$

According to *Lemma 1*, the temporal access-density consists of the number of newly arrived requests during the current prediction window and the decay of the temporal access-density of the previous prediction windows. Hence the temporal access-density captures that requests accessing a zone at different times have different effects on the prediction of the spatial locality of the zone.

3.2 Explore Sub-regional Spatial Locality

Depending on the temporal access-densities of the zones, we discuss the spatial locality of zones in the current prediction window. In a guest VM, the requests from background processes access some zones periodically. These zones do not have spatial locality, although their temporal access-densities are larger than zero. Therefore, the *TAC* algorithm uses a temporal access-density threshold to distinguish zones with future spatial locality. Considering the variations of the access frequencies of zones and interference of the system and journal processes in a VM, the temporal access-density threshold of a VM meet two conditions: (1) changing over time (*i.e.*, when updating the temporal access-densities of zones); and (2) never dropping suddenly when increasing the number of zones with low temporal access-densities. Thus the temporal access-density threshold of a VM in the current prediction window can be stated as the mean of the accessed zones' temporal access-densities that are larger than 1. By excluding zones whose temporal access-densities are lower than 1, we reduce the impacts of the zones that were accessed a long time ago, and therefore avoid any sudden drop of the threshold.

Definition 3: The temporal access-density threshold of a VM (*VM*) in the current prediction window is:

$$\delta(VM,T) = \sum_{y=1}^{N(T)} D(Z_y,T)/N(T) \quad (4)$$

where $N(T)$ is the number of zones whose $D(Z_y,T) \geq 1$.

Based on the current temporal access-density threshold of the VM, we explore the possibility of sub-regional spatial locality of VM. When $D(Z_j,\ T)$ is larger than $\delta(VM,\ T)$, the access of data in Z_j has sub-regional spatial locality in the next prediction window. Besides, the range of the sub-region with spatial locality may include zones with temporal access-densities more than $\delta(VM,T)$ and with temporal access-densities lower than $\delta(VM,T)$. For example, as shown in Fig. 3, in T_2, $D(Z_j,\ T_2)$ is larger than $\delta(VM,\ T_2)$ and $D(Z_{j+1},T_2)$ is smaller than $\delta(VM,T_2)$, but Z_{j+1} has sub-regional spatial locality in T_3. Therefore, when $D(Z_j,\ T)$ is larger than $\delta(VM,\ T)$, but the temporal access-density of Z_{j+1} is smaller than $\delta(VM,\ T)$, the *TAC* algorithm considers that both Z_j and Z_{j+1}[5] have sub-regional spatial locality of access in the next prediction window. Accordingly, the current distribution of the sub-regions with spatial locality consists of zones with higher temporal access-densities compared to the current temporal access-density threshold, and their neighbors with temporal access-densities lower than the current temporal access-density threshold.

3.3 Access Regularity of Sub-regional Spatial Locality

Given that different sub-regions have different access regularity and different zones may have different access regularity, we introduce a sub-region unit which comprises one or two neighboring zones with the same access regularity and relative spatial locality. Therefore, the range of a sub-region unit is defined as follows, in accordance with the above subsection on exploring sub-regional spatial locality:

Definition 4: The range of a sub-region unit in the current distribution of sub-regions with spatial locality of the VM is:

$$SR(U_i) = \begin{cases} (Z_j, Z_{j+1}); \\ \quad D(Z_j,T) \geq \delta(VM,T), D(Z_{j+1},T) < \delta(VM,T) \\ Z_j; \\ \quad D(Z_j,T) \geq \delta(VM,T), D(Z_{j+1},T) \geq \delta(VM,T) \end{cases} \quad (5)$$

where Z_{j+1} belongs to other sub-region units with relative spatial locality when $D(Z_{j+1},\ T) \geq \delta(VM,\ T)$.

According to *Definition 4*, the sub-region unit with spatial locality includes one zone whose temporal access-density is more than the temporal access-density threshold of the VM, and its neighbor with temporal access-density lower than this threshold. This allows us to gather arrival intervals between historical requests with relative sub-regional spatial locality to predict the arrival time interval of future requests. To reduce the cost of the model and remove the interference of discrete requests, we use the average access time interval of the zone with temporal access-density more than the threshold, in order to estimate the access time interval of the corresponding sub-region unit.

Definition 5: The future access interval of a sub-region unit with sub-regional spatial locality is:

$$ST(U_i) = ZT(Z_j, T(R_j^m)), D(Z_j,T) \geq \delta(VM,T) \quad (6)$$

5. Z_{j-1} is not included due to the following reasons :(1) reducing disk backward seek and rotation overheads; and (2) avoiding overlapping zones.

where $ZT(Z_j, T(R_j^m))$ is the average access interval of Z_j when R_j^m access Z_j, and is denoted by:

$$ZT(Z_j, T(R_j^m)) = \frac{ZT(Z_j, T(R_j^{m-1})) * (m-1) + T(R_j^m) - T(R_j^{m-1})}{m} \quad (7)$$

4 Disk Scheduling Based on Special Spatial Locality

Our aim is to exploit the special spatial locality of VM to improve the performance of *mixed* applications and disk I/O efficiency in virtualized environments while preserving *virtualization transparency*. Given that the non-work-conserving mode is good at exploiting spatial locality — it prefers to wait for incoming requests whose LBAs are closest to the position of disk head rather than dispatching pending requests and therefore avoid the disk seek overhead— and based on the observations in Section 2, our design needs to answer two critical questions: (1) whether to wait for future request and how long is the waiting time when being ready to dispatch request; (2) how to maximize disk I/O utilization while guaranteeing I/O performance of VMs, especially when VMs with strong sub-regional spatial locality and VMs with weak sub-regional spatial locality share storage.

We design and implement an adaptive non-work-conserving disk scheduling framework with a *spatial-locality-aware* (*SPLA*) heuristic algorithm in the hypervisor, named *Pregather*. The *SPLA* heuristic algorithm takes advantage of the regional spatial locality across VMs and the sub-regional spatial locality prediction of the *vNavigator* model, to guide *Pregather* to make the decision on waiting for future requests. Besides, based on the special spatial locality and the I/O characteristics of VMs, *Pregather* allocates the dynamic I/O service time to each VM, to reduce I/O contention among VMs while improving the disk I/O utilization.

Algorithm 1: Timer adjustment

Input:*P(LR)*: the LBA of the last completed request; $P(neighbor(VM_x).PR)$: the LBA of the pending request from a close neighbor VM_x; $Q(VM_x)$ the request queue of VM_x; U_i: the i^{th} sub-region unit of the *vNavigator* model; $size(VM_x)$: the size of VM_x image
Output:*coarseTimer or fineTimer is set*
/*make the decision on setting a timer after completing a request*/
if $Q(\text{VM}_x) == Null \&\& (AvgD(\text{VM}_x) < size(\text{VM}_x)$ $\&\& (AvgD(\text{VM}_x) < |P(neighbor(\text{VM}_x).PR) - P(LR)|)$ **then**
  $coarseTimer = AvgT(VM_x) + currentTime$
else if $Q(\text{VM}_x)! = Null \&\& P(LR) \in SR(U_i)$ **then**
  $fineTimer = ST(U_i) + currentTime$
end

4.1 Spatial-locality-aware Heuristic Algorithm

The key function of the *SPLA* heuristic algorithm is to evaluate the relationship between the cost of waiting for future requests and the cost of disk seeking for serving the pending request whose LBA is close to the position of disk head. To end this, the *SPLA* heuristic algorithm works considering the position of the disk head, the access regularity of the regional and sub-regional spatial locality of VMs, and the pending requests.

Algorithm 2: *SPLA* heuristic algorithm

Input:*PR*: the pending request; *P(LR)*; coarseTimer; fineTimer;U_i: the i^{th} sub-region unit corresponding to *P(LR)*;*newR*: a new request; $AvgT(VM_x)$
Output:*dispatch_req: a dispatching request*
/*make the decision when preparing to dispatch a request*/
begin
  $dispatch\_req = LP$
  if coarseTimer *is not over*$\&\&(PR \in \text{VM}_x \parallel AvgT(\text{VM}_x) \geq SeekTime(P(LR), P(PR)))$ **then**
    $dispatch\_req = PR$; turn off $coarseTimer$
  else if fineTimer *is not over* $\&\& P(PR) \in SR(U_i) \parallel ST(U_i) \geq SeekTime(P(LR), P(PR)))$ **then**
    $dispatch\_req = PR$; turn off $fineTimer$
  end
  /*Procedure invoked upon waiting for a future request*/
  while coarseTimer *or* fineTimer *is not over* && PR.deadline *is not over* $\&\&(dispatch\_req == LR)$ **do**
    /*on the arrival of the new request $newR$*/
    if $coarseTimer \&\& (newR \subseteq \text{VM}_x \parallel AvgT(\text{VM}_x) \geq SeekTime(P(newR), P(PR)))$ **then**
      $dispatch\_req = new\_req$; turn off $coarseTimer$
    end
    else if $(fine\_timer \&\& (P(newR) \in SR(U_i) \parallel ST(U_i) \geq SeekTime(P(newR), P(PR)))$ **then**
      $dispatch\_req = PR$; turn off $fineTimer$
    end
  end
  /*Procedure invoked upon expiration of coarse_timer or fine_timer or deadline of *PR**/
  if $(dispatch\_req == LR)$ **then**
    $dispatch\_req = PR$; turn off all timers
  end
end

After completing a request (*i.e.*, *LR*) from the current served VM[6] (*i.e.*, VM_x), *Pregather* introduces a timer to wait for future requests with special spatial locality (shown in Algorithm 1). According to the observations in Section 2, when all pending requests are not from the current served VM, exploiting the regional spatial locality can lower disk seek overheads across VMs. When some pending requests are from the current serving VM, exploiting the sub-regional spatial locality can reduce the disk seek overhead between sub-regions. So *Pregather* uses two types of timers — a coarse waiting time for future requests with regional spatial locality and a fine waiting time for future requests with sub-regional spatial locality— and then chooses the timer based on the position of the disk head and the current pending requests. If the hypervisor does not have any pending requests from VM_x after dispatching *LR*, *Pregather* considers whether to set the coarse waiting time: if the average distance between LBAs of requests from VM_x (*i.e.*, $AvgD(VM_x)$) is smaller than the size of VM image and the distance between the LBAs of *LR* and the pending request from a close neighbor VM, the coarse waiting time is set to the average arrival interval of VM_x (*i.e.*, $AvgT(VM_x)$). Instead, if VM_x has pending requests, *Pregather* decides whether to set the fine waiting time

6. The LBA of a completed request is the position of disk head at this time

and consults the *vNavigator* model to predict the sub-regional spatial locality around the disk-head position. If the model predicts that the position of *P(LR)* is in the range of a sub-region unit with spatial locality (*i.e.*, $SR(U_i)$), the fine waiting time is set to the average access interval of the sub-region unit (*i.e.*, $ST(U_i)$).

When the physical block device driver prepares to dispatch a request, as shown in Algorithm 2, *Pregather* selects the closest pending request (*i.e.*, *PR*) to *LR* according to the LBAs. *Pregather* estimates the disk seek time between the LBAs of *LR* and *PR* (*SeekTime(P(LR), P(PR))*). When the coarse waiting time is set, if *PR* meets one of the following two conditions: either the estimated seek time between the request and *P(LR)* is smaller than $AvgT(VM_x)$; or the request is from VM_x, the algorithm decides to dispatch *PR*. Otherwise, the algorithm decides to wait for a new request that meets one of these two conditions. On the other hand, when the fine waiting time is set, if LBA of *PR* also belongs to $SR(U_i)$, or *SeekTime(P(LR), P(PR))* is smaller than $ST(U_i)$, the algorithm decides to dispatch *PR*. Otherwise, the algorithm waits for a new request whose LBA belongs to $SR(U_i)$ or estimated seek time is smaller than $ST(U_i)$. Once the timer or the deadline of *PR* is overtime, *Pregather* dispatches *PR* to avoid from starving any requests.

4.2 Adaptive Time Slice Allocation among VMs

When VMs with strong special spatial locality and VMs with weak special spatial locality are run together, the *SPLA* heuristic algorithm, as it mainly focusses on exploiting special locality, speed up I/O performance of VMs with strong spatial locality at cost of the performance of VM with weak special spatial locality. This may lead to unfairness among VMs. Some schedulers rely on static time slice allocation for VMs, such CFQ combined with Blkio-Cgroup [17], and therefore ensure performance isolation and fairness among VMs. As discussed in [18], the actual bandwidth requirement of a workload is strongly related to the spatial locality and the size of request. These solutions based on the static time slice allocation don't consider the I/O characteristics of VMs, resulting in a waste of the disk I/O bandwidth). Accordingly, *Pregather* uses an adaptive I/O time slice allocation scheme for serving VMs and therefore avoids disk I/O contention and unfairness among VMs while keeping the efficiency of the *SPLA* heuristic algorithm.

The adaptive time slice allocation scheme initially assigns an equal time slice (*e.g.*, TS_x^0) to each VM, then adjusts the length of the time slice dynamically according to the spatial locality of VMs. In the virtualized environment, the size of requests is limited and always smaller than 44KB. So we ignore the impact of the request size on the time slice allocation. To quantize the special spatial locality of VM, we introduce a Regional spatial locality Indicator (*RI*, *e.g.* RI_x) and a Sub-regional spatial locality Indicator (*SI*, *e.g.*, SI_x) for each VM. The *RI* of a VM is stated as: $RI_x = AvgD(VM_x)/Sizeof(VM)$, where

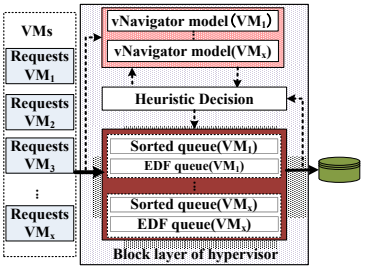


Fig. 4: **Architecture of *Pregather***

$AvgD(VM_x)$ is the average distance between LBAs of requests from VM and $Sizeof(VM_x)$ is the size of the VM image. The *SI* of a VM is the percentile of the disk seek distance between dispatched requests of VM at zero. When RI_x is larger than 1, this means that the VM does not own the regional spatial locality. So the length of the initial time slice is reduced, which leaves more time to serve the VMs with regional spatial locality. When RI_x is smaller than 1 then if SI_x is higher than 55%, the VM has clear sub-regional spatial locality and the length of the initial time slice of the VM is increased.

In the basis of the changes of both indicators, the length of the time slice of a VM (*e.g.*, TS_x) is changing as follows:

$$TS_x = \begin{cases} TS_x^0 \times (SI_x - 55\%) \times 10; & 0 < RI_x \leq 1, SI_x > 55\% \\ TS_x^0; & 0 < RI_x \leq 1, SI_x \leq 55\% \\ TS_x^0/2; & RI_x > 1 \end{cases} \tag{8}$$

According to (8), if RI_x is larger than 1, the length of time slice of VM_x is set to half of the length of the initial time slice, in order to avoid from wasting of disk I/O bandwidth while ensure the performance of this VM. Otherwise, if SI_x is smaller than 55%, the length of the time slice of VM_x is kept as the initial time slice to guarantee the performance of VMs with week sub-regional spatial locality. If SI_x is larger than 55%, the length of the time slice of VM is increased with the growth of the SI_x, to maximize the disk I/O utilization.

Furthermore, when the block layer does not have pending requests from a served VM and the time slice of the VM is not used up, the length of the time slice of the VM is reduced by half. This means that the type of requests from the served VM is not I/O intensive and *Pregather* gives more resources to VM with I/O intensive.

4.3 Implementation

We implement a prototype of *Pregather* in the Xen-hosted platform. As shown in Fig. 4, *Pregather* consists of the *vNavigator* models corresponding to VMs, and the *heuristic decision* module that implements the *SPLA* heuristic algorithm, at the block layer of the hypervisor. To make full use of the special spatial locality without starving requests, *Pregather*, similar to CFQ, builds two types of request queues for each VM: a sorted queue and an earlier deadline first (EDF) queue. The sorted queue contains requests in the order of their LBAs, while the EDF queue contains requests in the order of their deadline (the deadline value of *Pregather* is the same as of

CFQ). Also, *Pregather* allocates each VM a dynamic I/O time slice and serves VMs in a round robin fashion, to reduce I/O interference among VMs and batch as many requests from the same VM as possible. The round robin fashion decides the service order of VMs based on the distance between the position of VM and the position of disk head .

When a new request from a VM arrives in the hypervisor, *Pregather* assigns the request a deadline and queues the request in both the sort queue and EDF queue. Then *Pregather* calculate the average arrival time and the average distance between LBAs of requests from the VM, and updates the *RI* of the VM. Meanwhile, *Pregather* triggers the corresponding *vNavigator* model to analyze and update the range and arrival time interval of the sub-region units immediately.

After completing a request from the current served VM, the *heuristic decision* module sets a coarse timer or a fine timer based on the average arrival time and average distance of the VM and the *vNavigator* model. When the hypervisor prepares to dispatch a request, *Pregather* selects a pending request (named *PR*) before triggering the *heuristic decision* module. If the request queues of the current served VM are empty, *Pregather* selects a request from a VM whose location is close to the current served VM (in the disk). Otherwise, *Pregather* first checks the deadline of the head request in the EDF queue of the current served VM. If the deadline of the head request has expired, *Pregather* dispatches this request immediately. If not, *Pregather* selects a request next to the last completed request in the sorted queue of this VM as *PR*. Then, if the timer is active, the *heuristic decision* module decides whether to dispatch *PR*, without exceeding the deadline of *PR*. Once the *heuristic decision* module schedules a future request, *Pregather* maintains the idle state of the disk head until the arrival of a suitable request. If the timer runs out or the deadline of *PR* expires, *Pregather* dispatches *PR*.

Besides, after dispatching a request, *Pregather* computes the disk seek distance between the current dispatched request and the last dispatched request from the same VM. Then, *Pregather* updates the *SI* of the VM based on the disk seek distance. According to the change of both *RI* and *SI* and the number of pending requests of the VM, *Pregather* triggers the adaptive time slice allocation scheme to adjust the length of the time slice of the VM.

It is important to note that *Pregather* is not limited to Xen and can be implemented in other hypervisors (*e.g.*, KVM [19] and Linux-VServer [20]). The *vNavigator* model (introduced in section IV) uses hypervisor-independent parameters including the arrival times and LBAs of requests. Also, the *historical decision* module is implemented as a separate module at the block I/O layer of the hypervisor. Moreover, the *vNavigator* model can be applied in other physical block devices (*e.g.*, multiple disks, SSD), because the prediction of the model is not impacted by the physical characteristics of disks.

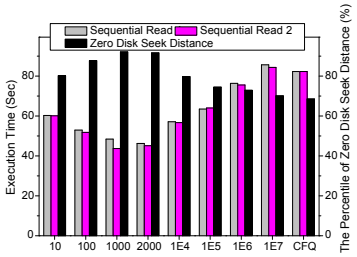


Fig. 5: **The execution time of two sequential read applications and the proportion of seek distance at zero point under *Pregather* with different offsets, CFQ**

5 Performance Evaluation

We run a suite of experiments evaluating our *Pregather* on the Xen-hosted platform using synthetic benchmarks, a MapReduce application and the database application. The first set of experiments is to verify the *vNavigator* model, including evaluation of the sensitive parameter and verification of its accuracy. The second set of experiments is to evaluate the overall performance of *Pregather* when there are multiple VMs with different access patterns. The third set of experiments is to evaluate the overheads caused by *Pregather*. The experimental setup is the same that described as in Section 2.1.

5.1 Verification of *vNavigator* Model

An accurate sub-regional spatial locality prediction is a key factor to achieve the high disk utilization with *Pregather*: *Pregather* uses the sub-regional spatial locality prediction of the *vNavigator* model to schedule requests. Hence we evaluate the accuracy of the prediction model.

The impact of B value. On the basis of the design of *vNavigator* model, the offset B of the model impacts the temporal access-densities of zones and the accuracy of clustering zones with spatial locality. Accordingly, to define a suitable B, we run two sequential read applications (16 threads sequentially read 128 files, whose total size is 2GB) in a VM, and capture their performance variation when changing B (we fix λ at 2, and the prediction window size to 20ms). The higher the accuracy of the prediction of the *vNavigator* model, the lower the frequency of disk seeking, because *Pregather* waits for a suitable future request with a minimal (zero) seek distance according to the *vNavigator* model. Thus we track the disk seek distance to discuss the prediction of the *vNavigator* model.

Fig. 5 shows the execution time of the two applications and the proportion of the disk seek distance at zero under different B values ($B = 1$ represents the size of a sector, *i.e.*, 512bytes). Increasing B from 10 to 1000 reduces the execution time of applications and increases the proportion of minimal seek distance. The reason for this is that small B leads to the temporal access-densities of all zones tending toward the same value. The *vNavigator* model treats a zone accessed by a background process as a zone with spatial locality, thus introducing unnecessary waiting, especially when B is equal to the size of the request. On the other hand,

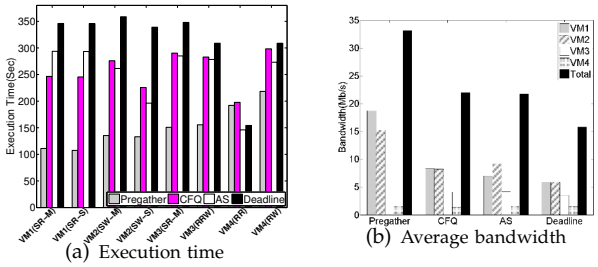
TABLE 3: **The Execution time of applications running on a VM**

VM	Workload	*Pregather*	CFQ	AS
SRRR VM	Sequential Read-M	26.92s	44.0s	45.83s
	Random Read	33.74s	44.75s	45.94
RRRR VM	Random Read	101.17s	119.18s	113.18s
	Random Read	108.29s	119.18s	112.39s
SWSW VM	Sequential Write-M	27.27s	40.07s	39.66s
	Sequential Write-M	28.04s	42.72s	39.35s
SWRW VM	Sequential Write-M	15.04s	25.79s	26.07s
	Random Write	47.06s	53.64s	53.05s
RWRW VM	Random Write	57.56s	66.39s	66.63s
	Random Write	57.56s	65.81s	66.63s
SWRR VM	Sequential Write-M	58.76s	81.10s	85.92s
	Random Read	81.78s	89.50s	88.80s
SRRW VM	Sequential Read-M	33.03s	44.06s	50.03s
	Random write	49.98s	52.75s	56.23s

when increasing B from 1000 to 2000, *Pregather* slightly decreases the average execution time of applications by 2%, but maintains 91.6% of the seek distance at zero. This is explained that the number of zones at 1000 is more than at 2000, introducing more time overheads when updating the temporal access-densities of zones. However, with increasing B from 2000 to 10^7, the execution time of both applications increases from 46s to 85s, while the proportion of seek distance at zero drops from 91.6% to 71.2%. The performance of *Pregather* at 10^7 is the same as that of CFQ. This is because the size of the zone may cover the complete disk region taken by the VM image with increasing B, which cause that our model cannot detect the possible sub-regional spatial locality. Therefore, B is restricted to the range between the size of the request and the size of the VM image.

The ratio of successful waiting. Based on the above discussion, we set B to 2000. Then, in seven different scenarios with different disk access patterns, we compare the performance of mixed applications within a VM under *Pregather*, CFQ and AS, to discuss the effectiveness of the model. Table 3 shows the execution time of the applications described in Table 1. The performance of applications under *Pregather* is better than under CFQ and AS. In particular, *Pregather* outperforms CFQ and AS by 33% and 31%, respectively, for the sequential write applications in the *SWSW* VM. By recording the activity of timer, we find that *Pregather* achieves a 90.6% success ratio on waiting for a suitable future request, and therefore reduces the seek time. In contrast, CFQ and AS treat the VM as a general process and thus never wait for the future request, although they also employ the non-work-conserving mode.

Moreover, the *vNavigator* model also captures the spatial locality of access between applications when sequential applications are mixed with random applications, and consequently improves the performance of the applications, as in *SRRR* VM, *SWRW* VM, *SWRR* VM, and *SRRW* VM. For instance, in *SRRR* VM, *Pregather* reduces the execution time of sequential read and random read by 38% and 22%, respectively, compared with CFQ and AS. In *SWRR* VM, *Pregather* outperforms CFQ for sequential write and random read applications by 27% and 9%, respectively. On the other hand, with *Pregather*, the improvement of random applications in *RWRW* VM and in *RRRR* VM is 12% and 10%, respectively. This is because the access pattern of random applications leads to the weak sub-regional spatial locality for a VM, as discussed in Section 2. Fortunately, the *vNavigator* model still prevents the interference of requests from background processes of a VM, and achieves an 80.4% success ratio on waiting for a suitable future request.



(a) Execution time (b) Average bandwidth

Fig. 6: **The performance of mixed applications when four VMs run together**

TABLE 4: **The distribution of disk seek distance when four VMs run together**

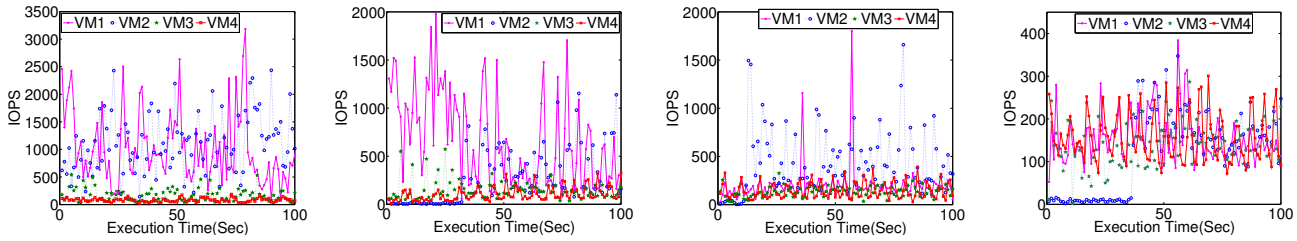
Distance	*Pregather*	CFQ	AS	Deadline
0	75.78%	55.06%	49.97%	12.54%
[-2.4E7,2.4E7]	97.39%	96.67%	96.57%	70.99%
[-1E9,-2.4E7]⋃ [2.4E7,1E9]	2.61%	3.37%	3.43%	29.01%

5.2 Spatial-Locality-Aware Disk Scheduling for Multiple VMs

We design experiments to measure the efficiency of *Pregather* for exploiting both the regional and sub-regional spatial locality (the initial I/O time slice for each VM is set to 200ms). Therefore, we compare the performance of multiple VMs under *Pregather* with that under three schedulers of the Xen-hosted platform (CFQ, AS and Deadline). Unlike CFQ and AS, Deadline is a work-conserving scheduler that dispatches neighboring requests without waiting for suitable future requests.

5.2.1 Mixed Applications with Different Access Patterns

We evaluate the performance of *Pregather* for the different mixed applications in the synthetic-benchmarks scenario described in Section 2.1. Fig. 6 shows the performance of mixed applications and VMs under four schedulers. Compared with CFQ, AS and Deadline, *Pregather* improves the bandwidth of VMs and performance of applications, especially VMs with sequential applications. The performance of Deadline as the representative of work-conserving mode is worse than other schedulers. Although both CFQ and AS use the non-working conserving mode to batch requests from the same VM, they do not achieve the high performance of applications because of ignoring the sub-regional spatial locality. For example, as shown in Fig. 6(a), for VM1 with different sequential applications, *Pregather* reduces the



(a) *Pregather* (b) CFQ (c) AS (d) Deadline

Fig. 7: **The change of throughput of VMs when four VMs with mixed applications run together**

execution time of both *SR-M* and *SR-S* applications by 55%, 62% and 68% in comparison with CFQ, AS and Deadline, respectively. Besides, the sub-regional spatial locality of VM3 is weaker than VM1 and VM2 as discussed in Section 2.2. Fortunately, *Pregather* boosts the performances of *SR-M* application and *RRW* application by more than 50% and more than 44% compared to the other schedulers. This is due to *Pregather* maximizing sub-regional spatial locality of VM3 in the I/O time slice of VM3. In addition, because of the limitation of physical bandwidth and the seeking overhead of the random applications, *Pregather* only improves the performance of the *RR* application and *RW* application in VM4 by 3% and 26% in comparison with CFQ. Although Deadline and AS outperform *Pregather* for the *RR* application in VM4, they sacrifice the performance of other VMs.

Fig. 6(b) also shows that *Pregather* improves the total bandwidth by 1.5x, 1.6x and 2.2x compared with CFQ, AS and Deadline, respectively. The VM4's bandwidth under *Pregather* is almost the same as that under Deadline and AS while the other VMs' bandwidth under *Pregather* is higher than the other schedulers. This is because (1)*Pregather* with the *SPLA* heuristic algorithm exploits the special spatial locality of the VM in a given time slice of the VM to improve disk I/O efficiency, (2)*Pregather* allocates the adaptive time slices to VMs based on the VMs' spatial locality, to ensure I/O performance of VMs.

Disk seek overheads. To further illustrate disk I/O efficiency with *Pregather*, Table 4 presents the distribution of the disk seek distance under four schedulers. The seek distance with *Pregather*, CFQ and AS is mainly concentrated in the range from -2.4×10^7 to 2.4×10^7, because the VM image occupies 2.4×10^7 sectors and the three schedulers exploit regional spatial locality across VMs to batch the requests from the same VMs. Besides, the proportion of disk seek distance at zero point with *Pregather* is 75.78%, higher than the other schedulers, because the *SPLA* heuristic algorithm for each VM continuously dispatches requests with the sub-regional spatial locality by waiting for future requests successfully. When the newly dispatched request and the last dispatched request are from different VMs between which the distance is far, or the newly dispatched request is from the journal process of the hypervisor, the seek distance is distributed in the range between -1×10^9 and -2.4×10^7 or from 2.4×10^7 to 1×10^9.

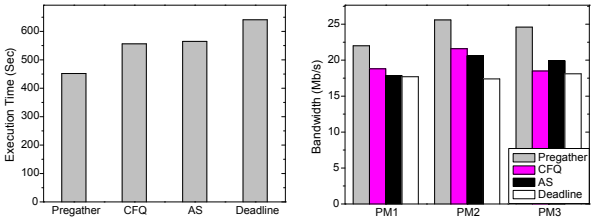
Time slice allocation among VMs. To discuss the efficiency of the adaptive time slice allocation in *Pregather*, Fig. 7 shows the changes in throughput of four VMs during 100s. As shown in Fig. 7(a), under *Pregather*, VMs achieve different throughput due to different degrees of their sub-regional spatial locality. For example, the throughput of both VM1 and VM2 is higher than that of VM3 and VM4, because VM1 and VM2 have stronger sub-regional spatial locality than VM3 and VM4. Meanwhile, although the sub-regional spatial locality of VM4 is weakest, the minimal throughput of VM4 is still more than *70IOPS*. Besides, the throughput of the VM2 often changes. For instance, the trend of VM2's throughput during 0 to 30s is different from its after 30s. This is because write operations lead to the changing sub-regional spatial locality and thus *Pregahter* adjusts the time slice of VM2 dynamically.

In contrast, Fig. 7(b) demonstrates that under CFQ, VM2 has slightly starved in the beginning when VM1 grabs a lot of bandwidth. Also, CFQ allocates a static and equal time slice to each VM and serves VMs in a round robin fashion, therefore it cannot guarantee the I/O performance of each VM and wastes the disk I/O resource due to ignoring the characteristics and the access patterns of VMs. Moreover, because both AS and Deadline neither assign a given I/O time slice to each VM nor serve VMs in the round robin fashion, as shown in the Fig. 7(c) and Fig. 7(d), the I/O interference among VMs is more serious and the throughput of VMs under AS and Deadline is lower than that under *Pregather*.

5.2.2 Real I/O Intensive Applications

To discuss the performance of *Pregather* for real I/O intensive applications, we test a Hadoop with sort benchmark under *Pregather*, CFQ, AS, and Deadline in two different scenarios: with and without a background VM hosting the database applications (TPC-H).

The experimental setup of the first scenario is described in Section 2.1. Fig. 8 shows the execution time and average bandwidth of each physical machine under the four schedulers. As shown in Fig. 8(b), *Pregather* improves the total bandwidth of the physical cluster by 26%, 28%, and 38% compared with CFQ, AS, and Deadline, respectively. Consequently, *Pregather* outperforms CFQ by 18% and AS by 20% for the sort benchmark. To discuss disk I/O efficiency further, Fig. 9 illustrates distribution of the seek distance in physical machines. Because Deadline sends as many adjacent requests as



(a) Execution time (b) Average bandwidth

Fig. 8: **The performance and throughput of VMs when running sort benchmark**

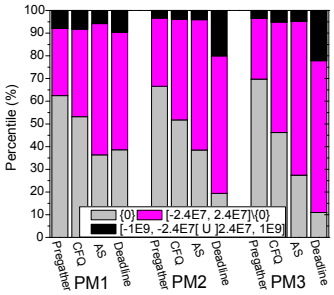
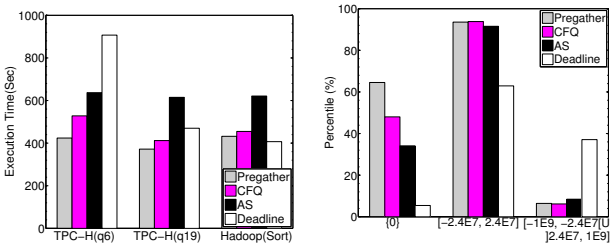


Fig. 9: **The distribution of seek distance of PMs when running sort benchmark**

possible without waiting for future requests, the seek overhead under Deadline is higher than that under the other schedulers, especially in PM2 and PM3 where only data nodes run. Although the distribution of the seek distance mainly concentrates on the space range of the image with *Pregather*, CFQ and AS for exploiting the spatial locality among VMs, the proportion of seek distances at zero with *Pregather* is much higher than with CFQ and AS. This is explained that the parallel processes of the sort benchmark sequentially access data sets during some periods, and thus leads to the clear sub-regional spatial locality of VMs (data nodes). *Pregather* captures these VMs' sub-regional spatial locality to reduce the disk seek overhead with the *SPLA* heuristic algorithm.

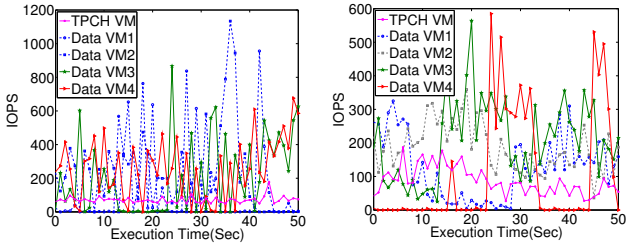
In the second scenario, we deploy a VM (named TPCH VM) with two different TPC-H instances (q6 and q19) and a five-nodes virtual cluster with Hadoop on a physical node. In the virtual cluster, we generate a 2GB data set in four VMs (named Data VM) each of which has two maps. Fig. 12 shows the execution time of the applications and the analysis of seek distance under *Pregather*, CFQ, AS and Deadline. We observe that *Pregather* improves the performance of three applications in contrast to CFQ and AS. For instance, compared with CFQ, *Pregather* reduces the execution time of sort benchmarks 20% because of capturing the clear sub-regional spatial locality of Data VMs. Meanwhile, *Pregather* only improves the performance of q6 and q9 by 9% and 4%, because the TPC-H instance accesses the data set randomly and TPCH VM own weak sub-regional spatial locality. Deadline costs the performance of Data VMs although improving the performance of TPCH VM. Besides, the proportion of minimal seek distance with *Pregather* is 65% and higher than that with other schedulers as shown in 10(b).

Furthermore, Fig. 11 describes the changes of Data



(a) Execution time (b) The distribution of seek distance

Fig. 10: **The performance and distribution of seek distance when the sort benchmark and TPC-H are running together**



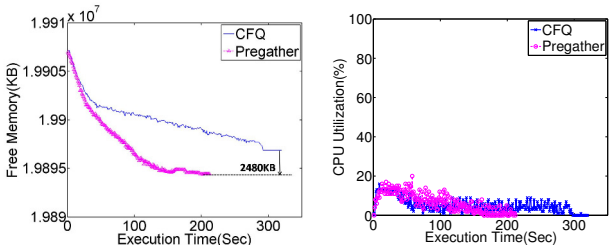
(a) Under *Pregather* (b) Under CFQ

Fig. 11: **The changes of throughput of Data VMs and TPC-H VM**

VMs' and TPCH VM's throughput under *Pregather* and CFQ during 100s, to discuss the time slice allocation of *Pregather* for the real and complicated I/O applications. As shown in Fig. 11(a), the changes of throughput in Data VMs are sharp, but their average throughput still achieves 300IOPS. Although the access patterns of processes and the arrival intervals of requests are variable in the sort benchmark as discussed in Section 2.3, *Pregather* can perceive the changes of sub-regional spatial locality of these VMs and then dynamically adjust the Data VMs' time slices. Besides, although the sub-regional spatial locality of TPCH VM is weaker, *Pregather* still maintains around 80IOPS the throughput of TPCH VM. In contrast, the TPCH VM grabs the bandwidth of Data VMs under CFQ. For example, as illustrated in Fig. 11(b), the highest throughput of TPC-H VM achieves 180IOPS and the average throughput is around 90IOPS. However, the average throughput of Data VMs with strong spatial locality are only 200IOPS and their jitter is serious.The reason is that CFQ neglects different degree of VMs' spatial locality and allocates the same time slice to VMs.

5.3 Overheads of *Pregather*

According to the implementation of *Pregather*, it stores the historical information of the accessed zones including the the temporal access-density, the number of arrived requests, the total arrival time interval and so on. Meanwhile, *Pregather* calculates the average distance of LBAs and access time interval between requests from the same VM, the temporal access-density, the average access interval of zones, and so on. Therefore, *Pregather* may cause the extra memory and CPU overheads. To evaluate these overheads, we compare the free memory and CPU utilization of the hypervisor under *Preagher* with those under CFQ in the synthetic-benchmarks sce-



(a) Free memory (b) CPU utilization

Fig. 12: **The memory and CPU overheads of hypervisor under *Pregather* and CFQ**

nario. This is because CFQ stores and computates the historical access information of VMs to predict the spatial locality of process.

As shown in Fig. 12(a), with the arrival of requests from VMs, the rate of decline of free memory under *Pregather* is faster than under CFQ. Compared with CFQ, *Pregather* uses less than 2148KB additional free memory. In contrast to the total free memory, the memory overheads caused by *Pregather* can be neglected. Fig. 12(b) shows that the highest CPU utilization under *Pregather* is no more than 20% and under CFQ achieves 17%. *Pregather* only introduces 3% of the extra CPU overheads compared to CFQ. Moreover, CPU utilization under *Pregather* and under CFQ are great low. So the I/O intensive applications do not spend CPU and the CPU power does not affect the performance of I/O applications, which are also discussed by Kundu et al [21].

6 Related Work

Ever since the advent of virtualization technology, a huge number of studies have been dedicated to improving the performance of disk-intensive applications in virtualized environments [4]–[9], [22]. On the one hand, some of these solutions use *invasive mode* scheduling to manage I/O requests [5], [6], [22]. They introduce an additional hypervisor-to-VM interface to achieve better coordination between the disk scheduler within both the hypervisor and VMs. However, these solutions can only be applied when VMs are running the same type of application. Moreover, they require the hypervisor to be aware of the applications running within VMs, which harms the transparency feature of virtualization.

On the other hand, some solutions use *non-invasive mode* scheduling to manage the I/O requests in virtualized environments without harming the transparency feature of virtualization [4], [7]–[9]. *Streaming scheduling* (SS) [9] turns any work-conserving disk scheduler into a non-work-conserving one based only on the request's own locality, thus reducing the disk seek time. SS essentially examines the existence of a stream by analyzing the characteristics of requests with relative spatial locality. Antfarm [4] enables the hypervisor to track the creation and exits of processes in VMs to infer the information of processes. The process information can help the disk scheduler at the hypervisor level to check the existence of read streams and map requests at the right read stream. However, both SS and Antfarm can only infer read streams, and cannot be applied for write applications or mixed applications running within a VM.

Besides, these solutions improve disk I/O throughput via capturing the regularity of read applications with strong spatial locality and speeding up their I/O performance. However, when multiple VMs with different applications run together, these solutions result in the disk resource contention among VMs and cannot guarantee I/O performance of VMs.

To the best of our knowledge, analyzing the access regularity of VMs has thus far been performed *only when a specific (one) application* is running within a VM (such as online transaction processing, mail server or file migration) [23]–[26]. Moreover, exploring the benefits of predicting the access regularity of processes to exploit the disk spatial locality (*e.g.*, by facilitating I/O perfecting) has so far been discussed only in *non-virtualized environments* [27]–[29]. These studies cannot be applied in virtualized environments, because unlike general processes, the access patterns of VM processes are more complicated and variable. Our work focuses on investigating and exploiting the disk access locality and regularity in *virtualized environments* when *mixed applications* are running within each VM. As far as we know, *we are the first to explore the benefits of the locality and regularity of data accesses to improve the disk efficiency in the presence of mixed applications while preserving the transparency feature of virtualization and ensuring I/O performance of VMs*.

7 Conclusion

In this study, we investigate the disk access patterns of VMs encapsulating mixed applications. Our studies reveal that disk accesses can be grouped into regions bounded by the virtual disks sizes, and within each region the disk accesses are grouped into sub-regions, which correspond to the applications' access patterns. Ignoring the special spatial locality (*i.e.*, the regional and sub-regional spatial locality) when scheduling I/O requests causes the performance degradation due to the high seek delay and the rotation overhead. We address this issue by developing *Pregather*, a new spatial-locality-aware disk scheduler that exploits the special spatial locality for improving disk-intensive applications. *Pregather* embraces an intelligent prediction model, named *vNavigator*, to predict the distribution of sub-regions within each region, and the arrival times of future requests accessing these sub-regions. We perform extensive experiments that involve multiple simultaneous applications of both synthetic benchmarks and a MapReduce application on Xen-based platforms. Our experiments show the accuracy of our prediction model and indicate that *Pregather* results in the high spatial locality and a significant improvement in disk throughput.

Regarding future work, to alleviate the lower spatial locality that occurs in the presence of disk fragmentswe

intend to extend *Pregather* to enable an intelligent allocation of physical blocks. Also, we are interested in applying *Pregather* in the data store environments consisting of multiple disks to improve I/O performance of VMs.

ACKNOWLEDGMENTS

The research is supported by National Science Foundation of China under grant No.61472151 and No.61232008, National 863 Hi-Tech Research and Development Program under grant No.2013AA01A208, Chinese Universities Scientific Fund under grant No.2013TS094, Research Fund for the Doctoral Program of MOE under grant No.20110142130005 and the ANR MapReduce grant (ANR-10-SEGI-001).

REFERENCES

[1] Amazon Web Services. http://aws.amazon.com/.
[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[3] D. Le, H. Huang, and H. Wang, "Understanding performance implications of nested file systems in a virtualized environment," in *Proc. FAST'12*, 2012, pp. 8–8.
[4] S. Jones, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Antfarm: Tracking processes in a virtual machine environment," in *Proc. ATC'06*, 2006, pp. 1–14.
[5] S. Ibrahim, H. Jin, L. Lu, B. He, and S. Wu, "Adaptive disk I/O scheduling for mapreduce in virtualized environment," in *Proc. ICPP'11*, 2011, pp. 335–344.
[6] M. Kesavan, A. Gavrilovska, and K. Schwan, "On disk I/O scheduling in virtual machines," in *Proc. WIOV'10*, 2010, pp. 6–6.
[7] S. Seelam and P. Teller, "Virtual I/O scheduler: a scheduler of schedulers for performance virtualization," in *Proc. VEE'07*, 2007, pp. 105–115.
[8] X. Ling, H. Jin, S. Ibrahim, W. Cao, and S. Wu, "Efficient disk I/O scheduling with qos guarantee for xen-based hosting platforms," in *Proc. CCGrid'12*, 2012, pp. 81–89.
[9] Y. Xu and S. Jiang, "A scheduling framework that makes any disk schedulers non-work-conserving solely based on request characteristics," in *Proc. FAST'11*, 2011, pp. 9–9.
[10] A. Kopytov, "Sysbench manual," http://sysbench.sourceforge.net/docs/.
[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.
[12] J. Axboe and A. D. Brunelle, "Blktrace User Guide," http://www.cse.unsw.edu.au/aaronc/iosched/doc/blktrace.html.
[13] The QCOW Image Format. http://people.gnome.org/ markmc/qcow-image-format-version-1.html.
[14] C. Tang, "FVD: a high-performance virtualmachine image format for cloud," in *Proc. USENIX ATC'11*, 2011, pp. 18–18.
[15] J. Axboe. Completely Fair Queuing (CFQ). http://en.wikipedia.org/wiki/CFQ, 2010.
[16] S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O," *ACM SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 117–130, 2001.
[17] blkio-contoller. https://www.kernel.org/doc/Documentation/cgroups/blkio-controller.txt.
[18] X. Zhang, K. Davis, and S. Jiang, "Qos support for end users of i/o-intensive applications using shared storage systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 18.
[19] KVM. http://www.linux-kvm.org/page/mainpage.
[20] Linux VServer. http://linux-vserver.org/Documentation, 2010.
[21] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," *Proc. of VEE'12*, vol. 47, no. 7, pp. 3–14, 2012.
[22] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passé?" *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 20–24, Mar. 2010.
[23] Y. Hu, X. Long, and J. Zhang, "I/O behavior characterizing and predicting of virtualization workloads," *Journal of Computers*, vol. 7, no. 7, pp. 1712–1725, 2012.
[24] I. Ahmad, "Easy and efficient disk I/O workload characterization in vmware esx server," in *Proc. IISWC'07*, 2007, pp. 149–158.
[25] I. Ahmad, J. Anderson, A. Holler, R. Kambo, and V. Makhija, "An analysis of disk performance in VMware ESX server virtual machines," in *Proc. WWC'03*, 2003, pp. 65–76.
[26] A. Gulati, C. Kumar, and I. Ahmad, "Storage workload characterization and consolidation in virtualized environments," in *Proc. VPACT'09*, 2009.
[27] N. Tran and D. Reed, "Automatic arima time series modeling for adaptive I/O prefetching," *IEEE TPDS*, vol. 15, no. 4, pp. 362–377, 2004.
[28] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang, "Diskseen: exploiting disk layout and access history to enhance I/O prefetch," in *Proc. USENIX ATC'07*, 2007, pp. 1–14.
[29] Z. Li, Z. Chen, S. Srinivasan, and Y. Zhou, "C-miner: Mining block correlations in storage systems," in *Proc. FAST'04*, vol. 186, 2004.

Xiao Ling is an engineer at Information & Communication Company of Hunan Electric Power Corporation of State Grid in China. She received her B.S. (2008) degree at National University of Defense Technology, and M.S. (2011) and Ph.D. (2014) degree at Huazhong University of Science and Technology. Her research interests are in the area of cloud computing and virtualization, focusing on virtualized I/O optimization in cloud.

Shadi Ibrahim is a researcher within the KerData team at INRIA Rennes, working on scalable distributed data processing in the Cloud. He holds a Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST) in China. He has several years of experience with academic research at HUST, INRIA research center and Microsoft research center in Asia. His research interests include cloud computing, data-intensive computing, virtualization technology, file and storage systems.

Song Wu is a professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. He received his Ph.D. from HUST in 2003. He is now the director of Parallel and Distributed Computing Institute at HUST. He is also served as the vice director of Service Computing Technology and System Lab (SCTS) and Cluster and Grid Computing Lab (CGCL) of HUST. His current research interests include grid/cloud computing and virtualization technology.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in China. He is now Dean of the School of Computer Science and Technology at HUST. Jin received his PhD in computer engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System. Jin is a senior member of the IEEE and a member of the ACM. Jin is the member of Grid Forum Steering Group (GFSG). He has co-authored 15 books and published over 400 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security.